CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CH-3 Array & String Handling

DEPSTAR

Department of Computer Engineering

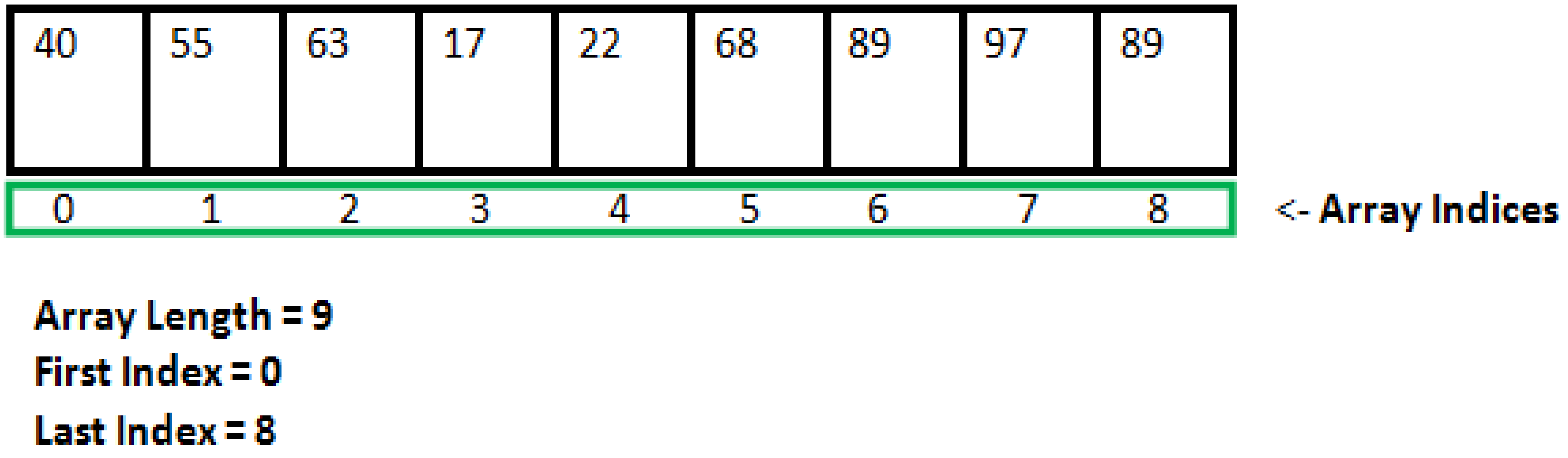
Devang Patel Institute of Advance Technology and Research

Charotar University of Science and Technology

Contents

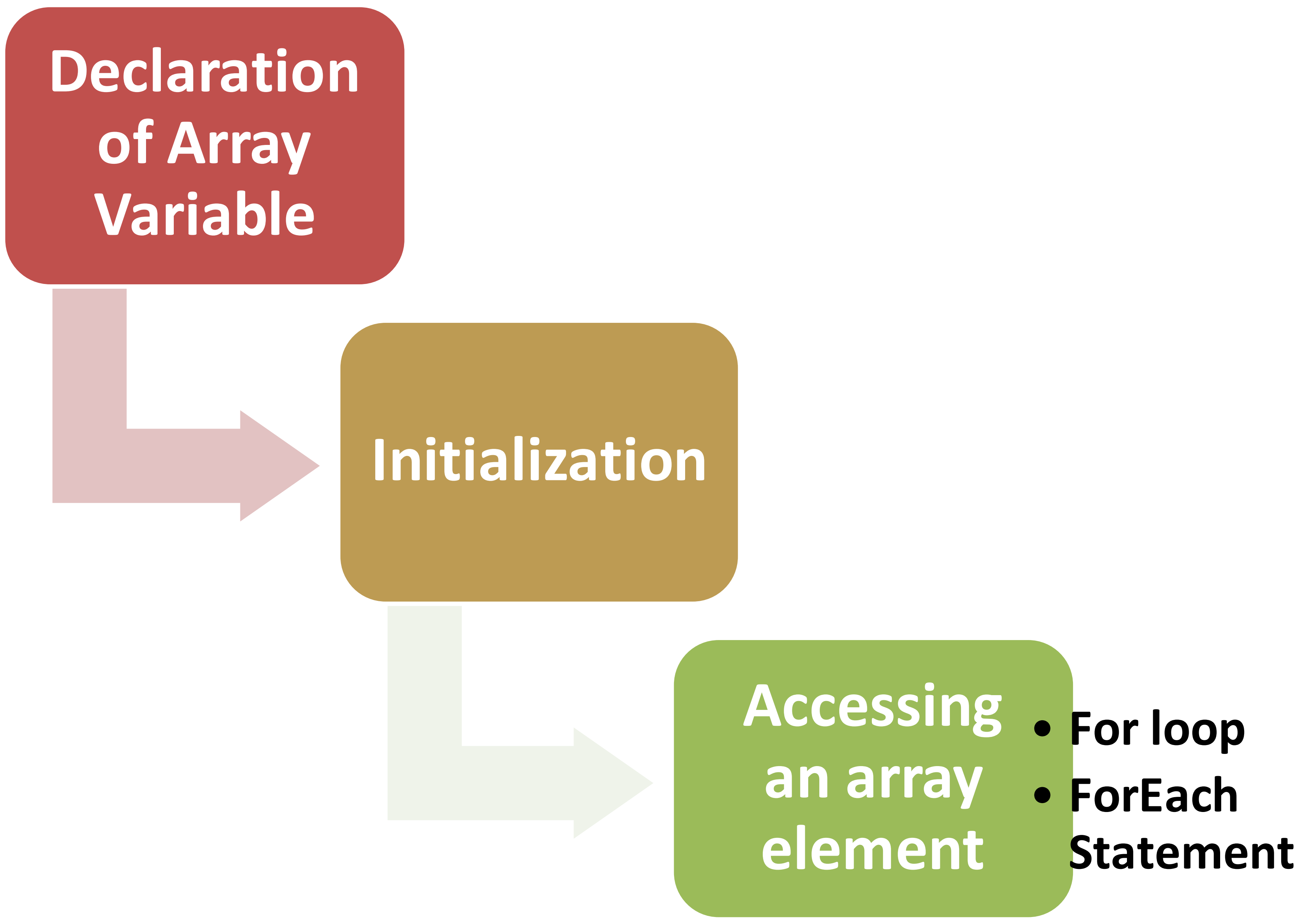
Array Basic

- An array is a group of like-typed variables that are referred to by a common name
- Arrays in Java work differently than they do in C/C++



Limitations of Array in JAVA

- Standard Java arrays are of a **fixed length**.
- After arrays are created, they **cannot grow or shrink**, which means that you must know in advance how many elements an array will hold.



- For loop
- ForEach Statement

Declaration of the array

- type var-name[ ]; or typep[] var-name;

- Example:
  - byte[] anArrayOfBytes;
  - short[] anArrayOfShorts;
  - long[] anArrayOfLongs;
  - float[] anArrayOfFloats;
  - double[] anArrayOfDoubles;
  - boolean[] anArrayOfBooleans;
  - char[] anArrayOfChars;
  - String[] anArrayOfStrings;
  - float anArrayOfFloats[];

Creation of array

- For example consider the declaration :

  int month_days[];

- Declaration establishes the fact that month_days is an array variable, no array actually exists.
- To link **month_days** with an actual, physical array of integers, you must allocate one using **new** and assign it to **month_days**.
- **new** is a special operator that allocates memory.

array-var = new type [size];

• *array-var = new type [size];*

- ***type*** specifies the type of data being allocated, *size* specifies the number of elements in the array,
- ***array-var*** is the array variable that is linked to the array.
- **new** is use to allocate an array, you must specify the type and number of elements to allocate

- The elements in the array allocated by **new** will automatically be initialized to zero (for numeric types), **false** (for **boolean**),

- This example allocates a 12-element array of integers and links them to month_days:

month_days = new int[12];

- **Obtaining an array is a two-step process.**
  - **First, you must declare a variable of the desired array type.**
  - **Second, you must allocate the memory that will hold the array, using new, and assign it to the array variable.**
- Thus, in Java all arrays are **dynamically allocated**.

Initialization of Array

- Once you have allocated an array, you can access a specific element in the array by specifying its index within square brackets.
- All array indexes start at zero.
- For example, this statement assigns the value 28 to the second element of **month_days**:
- **month_days[1] = 28**;

Accessing Array elements

```
// Demonstrate a one-dimensional array.

public class Array {
public static void main(String args[])
        {
        int month_days[];
        month_days = new int[12];
        month_days[0] = 31;
        month_days[1] = 28;
        month_days[2] = 31;
        month_days[3] = 30;
        month_days[4] = 31;
        month_days[5] = 30;
        month_days[6] = 31;
        month_days[7] = 31;
        month_days[8] = 30;
        month_days[9] = 31;
        month_days[10] = 30;
        month_days[11] = 31;
        System.out.println("April has " + month_days[3] + " days.");
        }
}
```

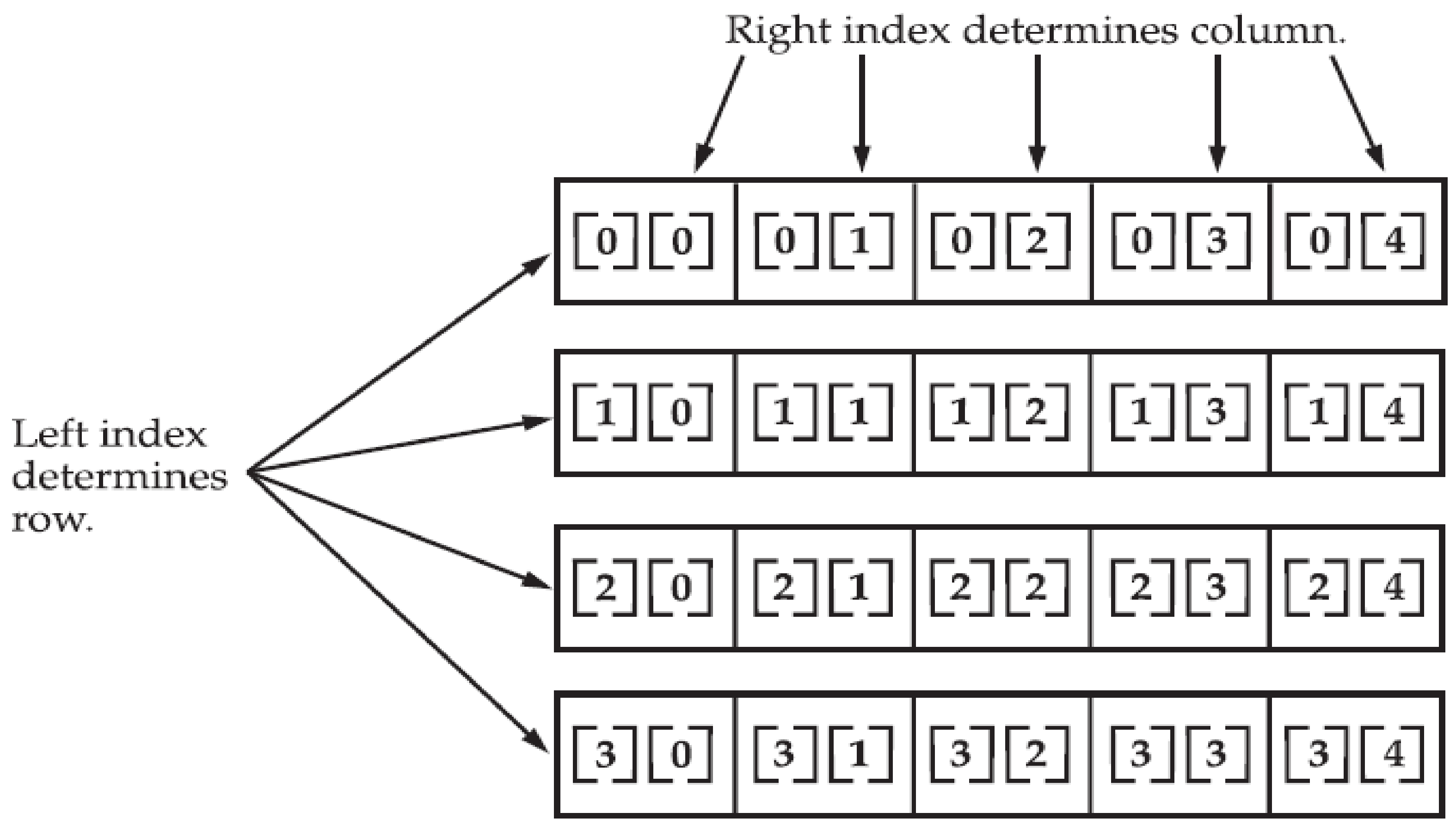
```
public class Main {
        public static void forDisplay(int[] a)
                {
                        System.out.println("Display an array using for loop");
                        for (int i = 0; i < a.length; i++)
                         {
                                System.out.print(a[i] + " ");
                         }
                        System.out.println();
                }
         public static void foreachDisplay(int[] data)
                 {
                        System.out.println("Display an array using for each loop");
                        for (int a  : data)
                          {
                                System.out.print(a+ " ");
                               }
                       }
public static void main(String[] args)
                {
                        int[] intary = { 1,2,3,4};
                        forDisplay(intary);
                        foreachDisplay(intary);
                }

}
```

Multidimensional Arrays

- Arrays of arrays

```
int twoD[][] = new int[4][5];
```



```
// Demonstrate a two-dimensional array.
class TwoDArray {
public static void main(String args[])
        {
        int twoD[][]= new int[4][5];
        int i, j, k = 0;
        for(i=0; i<4; i++)
                for(j=0; j<5; j++)
                {
                twoD[i][j] = k;
                k++;
                }
        for(i=0; i<4; i++)
                {
                for(j=0; j<5; j++)
                System.out.print(twoD[i][j] + " ");
                System.out.println();
                }
        }
}
```

```
// Demonstrate a two-dimensional  array.
import java.util.*;
class TwoDArrayIn {
public static void main(String args[])
            {
            int twoD[][]= new int[4][];
            twoD[0]=new  int[5];
            twoD[1]=new  int[5];
            twoD[2]=new  int[5];
            twoD[3]=new  int[5];

            Scanner  s=new Scanner(System.in);
            int i, j;
            System.out.println("Enter the Elements");
            for(i=0; i<4; i++)
                        for(j=0; j<5; j++)
                        {
                        twoD[i][j]  = s.nextInt();

                        }
            for(i=0; i<4; i++)
                        {
                        for(j=0; j<5; j++)
                        System.out.print(twoD[i][j]  + " ");
                        System.out.println();
                        }
            }
}
```

```
// Manually allocate differing size second dimensions.
class TwoDAgain {
public static void main(String args[])
          {
          int twoD[][] = new int[4][];
          twoD[0] = new int[1];
          twoD[1] = new int[2];
          twoD[2] = new int[3];
          twoD[3] = new int[4];

          int i, j, k = 0;
          for(i=0; i<4; i++)
                    for(j=0; j<i+1; j++)
                    {
                    twoD[i][j] = k;
                    k++;
                    }
          for(i=0; i<4; i++)
                    {
                    for(j=0; j<i+1; j++)
                    System.out.print(twoD[i][j] + " ");
                    System.out.println();
                    }
          }
}
```

// Initialize a two-dimensional array.

```
class Matrix {
public static void main(String args[])
        {
        int m[][] =
                {
                        { 0*0, 1*0, 2*0, 3*0 },
                        { 0*1, 1*1, 2*1, 3*1 },
                        { 0*2, 1*2, 2*2, 3*2 },
                        { 0*3, 1*3, 2*3, 3*3 }
                };
        int i, j;
        for(i=0; i<4; i++) {
        for(j=0; j<4; j++)
        System.out.print(m[i][j] + " ");
        System.out.println();
        }
        }
}
```

```
// Demonstrate a three-dimensional array.
class ThreeDMatrix {
public static void main(String args[])
          {
          int threeD[][][] = new int[3][4][5];
          int i, j, k;
          for(i=0; i<3; i++)
                    {
                    for(j=0; j<4; j++)
                              {
                              for(k=0; k<5; k++)
                              threeD[i][j][k] = k;
                              }
                    }

          for(i=0; i<3; i++)
                    {
                    for(j=0; j<4; j++)
                              {
                              for(k=0; k<5; k++)
                              System.out.print(threeD[i][j][k] + " ");
                              System.out.println();
                              }
                    System.out.println();
                    }
          }
}
```

StringArray

- A Java String Array is an object that holds a fixed number of String values.
- Arrays in general is a very useful and important data structure that can help solve many types of problems.

String Array Declaration

- String[] thisIsAStringArray;
- String thisIsAStringArray[];

String Array Declaration With Initial Size

- String[] thisIsAStringArray = new String[5];
- thisIsAStringArray[0] = "AAA";
- thisIsAStringArray[1] = "BBB";
- thisIsAStringArray[2] = "CCC";
- thisIsAStringArray[3] = "DDD";
- thisIsAStringArray[4] = "EEE“;

Accessing the elements

- System.out.println(thisIsAStringArray[0]);
  System.out.println(thisIsAStringArray[1]);
  System.out.println(thisIsAStringArray[2]);
  System.out.println(thisIsAStringArray[3]);
  System.out.println(thisIsAStringArray[4]);

```
public class StringArray {
        public static void main(String args[]) {
                // declare a string array with initial size
                String[] schoolbag = new String[4];
                // add elements to the array
                schoolbag[0] = "Books";
                schoolbag[1] = "Pens";
                schoolbag[2] = "Pencils";
                schoolbag[3] = "Notebooks";
                // this will cause ArrayIndexOutOfBoundsException
                // schoolbag[4] = "Notebooks";
                // declare a string array with no initial size
                // String[] schoolbag;
                // declare string array and initialize with values in one step
                String[] schoolbag2 = { "Books", "Pens", "Pencils", "Notebooks" };
                // print the third element of the string array
                System.out.println("The third element is: " + schoolbag2[2]);
                // iterate all the elements of the array
                int size = schoolbag2.length;
                System.out.println("The size of array is: " + size);
                for (int i = 0; i < size; i++) {
                        System.out.println("Index[" + i + "] = " + schoolbag2[i]);
                }
                // iteration provided by Java 5 or later
                for (String str : schoolbag2) {
                        System.out.println(str);
                }
        }
}
```

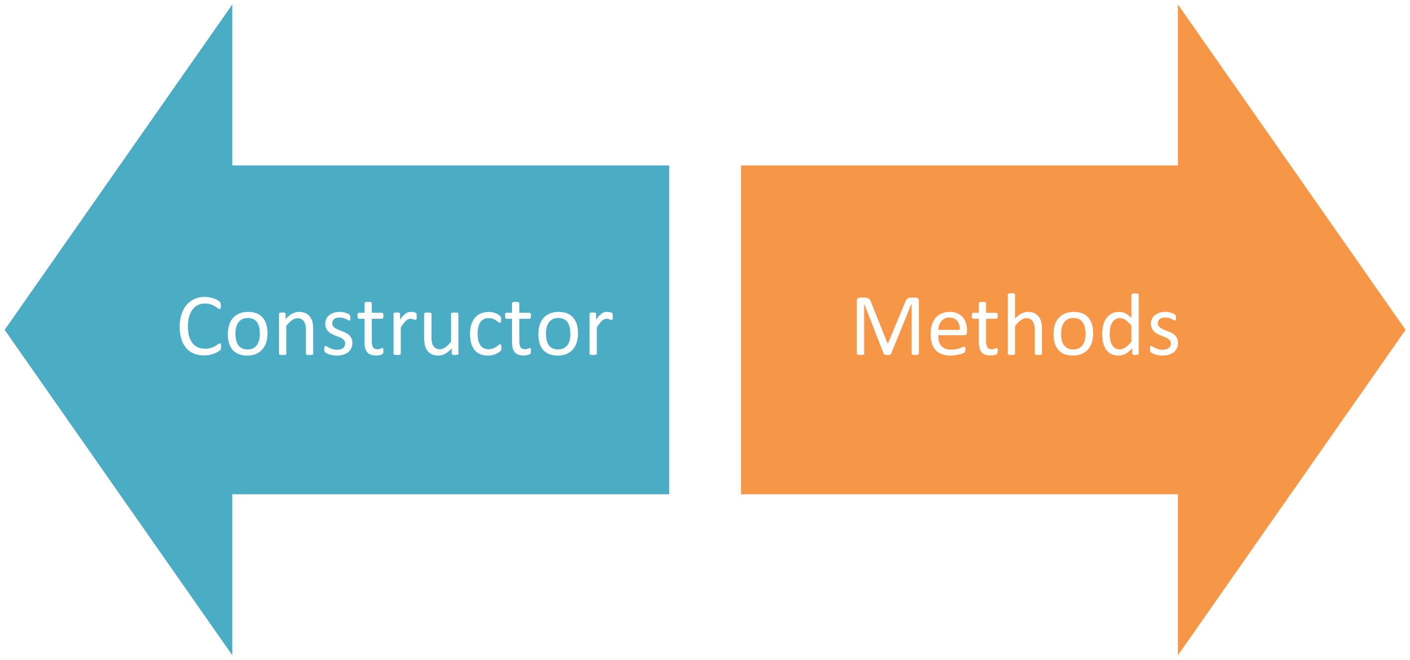
String Class

- Strings, which are widely used in Java programming, are a sequence of characters.
- In Java programming language, strings are treated as objects
- The most direct way to create a string is to write

```
String greeting = "Hello world!";
```

- Whenever it encounters a string literal in your code, the compiler creates a String object with its value in this case, "Hello world!'.

String Class



String Class-Constructors

1. **String (string literal)**

   String strl =new String("Java");

   When this statement is compiled, the Java compiler invokes this constructor and initializes the String object with the string literal enclosed in parentheses which is passed as an argument.

2. **String (charArray)**

   String strl = new String(chrArr);

   char[] charArray ={'H','i',' ','D','I','N','E','S','H'};

   You can also create a string from an array of characters. To create a string initialized by an array of characters, use the constructor of the form

3. String ()

String s = new String();

It constructs a new String object which is initialized to an empty string (" ")

4. String (String strObj)

String s2 = new String(s);

It constructs a String object which is initialized to same character sequence as that of the string referenced by strObj. For example: A string reference variable s references a String object which contains a string Welcome On execution of the above statement

5. String (char [] chArr, int startIndex, int count)

```
char[] str = {'W','e','l','c','o','m','e'};
String s3 = new String(str,5,2};
```

It constructs a new String object whose contents are the same as the character array, chArr, starting from index startIndex upto a maximum of count characters.

On execution, the string reference variable s3 will refer to a string object containing a character sequence **me** in it.

6. String (byte [] bytes)

```
byte[] ascii ={65,66,67,68,70,71,73};
String s4 = new string (ascii};
```

It constructs a new String object by converting the bytes array into characters using the default charset.

On execution, the string reference variable s4 will refer to a String object containing the character equivalent to the one in the bytes array

7. String (byte [] bytes, int startIndex, int count)

```
byte[] ascii ={65,66,67,68,70,71,73};
String s5 = new String(ascii,2,3);
```

It constructs a new String object by converting the bytes array starting from index startIndex upto a maximum of count bytes into characters using the default charSet

The string reference variable s5, will refer to a String object containing character sequence

```
public class StringConstructors
{
      public static void main(String[]  args)
   {
         char[] charArray ={'H','i',' ','D','I','N','E','S','H'};
         byte[] ascii ={65,66,67,68,70,71,73};
         String str  = "Welcome";
         String strl =new  String("Java");
         String str2 =new String(charArray);
         String str3 =new String(charArray,3,3);
         String str4 =new String(ascii);
         String str5 =new String(ascii,2,3);
         String str6 =new String();
         String str7 =new String(str);
         System.out.println("str  : "+ str);
         System.out.println("strl  : "+ strl);
         System.out.println("str2  : "+ str2);
         System.out.println("str3  : "+ str3);
         System.out.println("str4  : "+ str4);
         System.out.println("str5  : "+ str5);
         System.out.println("str6  : "+ str6);
         System.out.println("str7  : "+ str7);
         str = str+" Dinesh Thakur";
         System.out.println("str  : "+ str);
 }   }
```

String Class-Methods

1. char charAt(int index)

Returns char value for the particular index

- The index number starts from 0 and goes to n-1, where n is length of the string.
- It returns **StringIndexOutOfBoundsException** if given index number is greater than or equal to this string length or a negative number.

- **Signature**: **public char charAt(int index)**
- **Parameter:**

  Index : Index number, starts with 0
- **Returns: A char value**
- **Throws: StringIndexOutOfBoundsException : If index is negative value or greater than this string length.**

```
public class CharAtExample{
public static void main(String args[])
            {
            String name="javatpoint";
            char ch=name.charAt(4);//returns  the char value at the 4th index
            System.out.println(ch);
            }
}
```

```
public class CharAtException{
public static void main(String args[])
            {
            String name="javatpoint";
            char ch=name.charAt(10);//returns  the char value at the 10th index
            System.out.println(ch);
            }
}
```

```
public class CharAtExample3 {
    public static void main(String[] args) {
    String str = "Welcome to Javatpoint portal";
    int strLength = str.length();
    // Fetching first character
    System.out.println("Character at 0 index is: "+ str.charAt(0));
    // The last Character is present at the string length-1 index
    System.out.println("Character at last index is: "+
str.charAt(strLength-1));
    }
}
```

```
public class CharAtExample4 {
    public static void main(String[] args)
             {
        String str = "Welcome to Javatpoint portal";
        for (int i=0; i<=str.length()-1; i++)
                                    {
            if(i%2!=0)
                System.out.println("Char at "+i+" place "+str.charAt(i));


                                    }
    }
}
```

2. Java String length()

Returns string lengths.

The java string length() method length of the string. It returns count of total number of characters.

- **Signature**: **public int length()**
- **Returns: length of characters**

```
public class LengthExample{
public static void main(String args[])
            {
            String s1="javatpoint";
            String s2="python";
            System.out.println("string length is: "+s1.length());//10 is the length of
javatpoint string
            System.out.println("string length is: "+s2.length());//6 is the length of
python string
            }
}
```

```
public class LengthExample2 {
    public static void main(String[] args)
                {
        String str = "Javatpoint";
        if(str.length()>0)
                                            {
           System.out.println("String is not empty and length is: "+str.length());
                                            }
        str = "";
        if(str.length()==0)
                                            {
            System.out.println("String is empty now: "+str.length());
                                            }
    }
}
```

3. String format(String format, Object... args)

Returns returns formatted string

The java string format() method returns the formatted string by given locale, format and arguments.

- Signature: public static String format(String format, Object... args)
- Parameter:

  locale : specifies the locale to be applied on the format() method.

  format : format of the string.

  args : arguments for the format string. It may be zero or more.
- Returns: Formatted string
- Throws:
  - NullPointerException : if format is null.
  - IllegalFormatException : if format is illegal or incompatible.

```
public class FormatExample{
public static void main(String args[])
            {
            String name="sonoo";
            String sf1=String.format("name is %s",name);
            String sf2=String.format("value is %f",32.33434);
            String sf3=String.format("value is %32.12f",32.33434);//returns 12
char fractional part filling with 0

            System.out.println(sf1);
            System.out.println(sf2);
            System.out.println(sf3);
            }
}
```

```
public class FormatExample2 {
   public static void main(String[] args)
           {
      String str1 = String.format("%d", 101);          // Integer value
      String str2 = String.format("%s", "Amar Singh"); // String value
      String str3 = String.format("%f", 101.00);       // Float value
      String str4 = String.format("%x", 101);          // Hexadecimal value
      String str5 = String.format("%c", 'c');          // Char value
      System.out.println(str1);
      System.out.println(str2);
      System.out.println(str3);
      System.out.println(str4);
      System.out.println(str5);
   }
 }
```

3. String substring(int beginIndex, int endIndex)

Returns substring for given begin index and end index

We pass begin index and end index number position in the java substring method where start index is inclusive and end index is exclusive

- Signature: public String substring(int startIndex, int endIndex)
- Parameter:

  startIndex : starting index is inclusive

  endIndex : ending index is exclusive.
- Returns: specified string
- Throws:
  - **StringIndexOutOfBoundsException** : If start index is negative value or end index is lower than starting index

```
public class SubstringExample{

public static void main(String args[])

          {

          String s1="javatpoint";

          System.out.println(s1.substring(2,4));//returns va

          System.out.println(s1.substring(2));//returns vatpoint

          }

}
```

```
public class SubstringExample2 {
    public static void main(String[] args) {
        String s1="Javatpoint";
        String substr = s1.substring(0); // Starts with 0 and goes to end
        System.out.println(substr);
        String substr2 = s1.substring(5,10); // Starts from 5 and goes to 10
        System.out.println(substr2);
        String substr3 = s1.substring(5,15); // Returns Exception
    }
}
```

4. public int compareTo(String anotherString)

Compares two strings lexicographically.

- Lexicographically means : If two strings are different, then either they have different characters at some index that is a valid index for both strings, or their lengths are different, or both.
- If they have different characters at one or more index positions, let k be the smallest such index. In this case, compareTo returns the difference of the two character values at position k in the two string .
- If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string. In this case, compareTo returns the difference of the lengths of the strings.

- **Signature**: **public int compareTo(String anotherString)**
- **Parameter: another string.**
- **Returns: zero, positive or negative value**

No.	Method	Description
1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
3	static String format(String format, Object... args)	returns formatted string
4	static String format(Locale l, String format, Object... args)	returns formatted string with given locale
5	String substring(int beginIndex)	returns substring for given begin index
6	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index
7	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value
8	static String join(CharSequence delimiter, CharSequence... elements)	returns a joined string
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	returns a joined string
10	boolean equals(Object another)	checks the equality of string with object

11	boolean isEmpty()	checks if string is empty
12	String concat(String str)	concatinates specified string
13	String replace(char old, char new)	replaces all occurrences of specified char value
14	String replace(CharSequence old, CharSequence new)	replaces all occurrences of specified CharSequence
15	static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
16	String[] split(String regex)	returns splitted string matching regex
17	String[] split(String regex, int limit)	returns splitted string matching regex and limit
18	String intern()	returns interned string
19	int indexOf(int ch)	returns specified char value index
20	int indexOf(int ch, int fromIndex)	returns specified char value index starting with given index
21	int indexOf(String substring)	returns specified substring index
22	int indexOf(String substring, int fromIndex)	returns specified substring index starting with given index
23	String toLowerCase()	returns string in lowercase.
24	String toLowerCase(Locale l)	returns string in lowercase using specified locale.
25	String toUpperCase()	returns string in uppercase.
26	String toUpperCase(Locale l)	returns string in uppercase using specified locale.
27	String trim()	removes beginning and ending spaces of this string.
28	static String valueOf(int value)	converts given type into string. It is overloaded.